CMPT 280: Intermediate Data Structures and Algorithms

Sample Final Exam Questions

data, operation, encapsulation

Dislcaimer: These are some samples of questions that have appeared on past final exams. It is not an exhaustive sampling of questions you might encounter on this year's exam. Other types of questions might occur. Remember also that about 50% of the final exam will be multiple choice questions.

1. (a) (3 points) What are the three components that an Abstract Data Type has, regardless of the method of ADT specification used? *(Note: not looking for "sets, syntax, semantics..." here, those are components of a specific ADT specification method.)*

   (b) (5 points) Suppose you are designing data structure for storing a list of legal words to be used in a spell-checking application. For each of the following, indicate whether they are decisions that should be made at the time of ADT Specification, or ADT Implementation and give a few words (no more than one short sentence) of justification each of your answers:

   - Decide whether to store the words in a linked list or a trie
   - Decide what characters are allowed to appear in a word
   - Decide whether words are allowed to be deleted from the dictionary
   - Decide whether testing for equality of words is case-sensitive
   - Decide what language to use to write the spell-checker

2. (10 points) In lib280, the `SimpleList280` class has the following methods:

```
public boolean isEmpty();

public boolean isFull();

public void clear();

/**   Insert x as the first element in the list.
   @precond !isFull()
   @param x item to be inserted in the list */
public void insertFirst(I x) throws ContainerFull280Exception;

/**   Return the first item.
   @precond !isEmpty()
*/
public I firstItem() throws ContainerEmpty280Exception;

/**   Delete the first item from the list.
   @precond    !isEmpty()
*/
public void deleteFirst() throws ContainerEmpty280Exception;
```

   Using the *white-box testing* method for generating test cases, generate 10 test cases that could be used for regression testing of `SimpleList280`. The set of 10 tests need not be exhaustive, but should consist of valid, reasonable test cases. Do not write the code for the test cases, just list the test cases, e.g. "test whether `isEmpty()` returns true when the list is empty".

3. (5 points) What are the worst-case time complexities of the following algorithms? Give your answers in Big-O notation, and define your variables, e.g. "Binary search is $O(\log n)$ where n is the number of elements being searched."

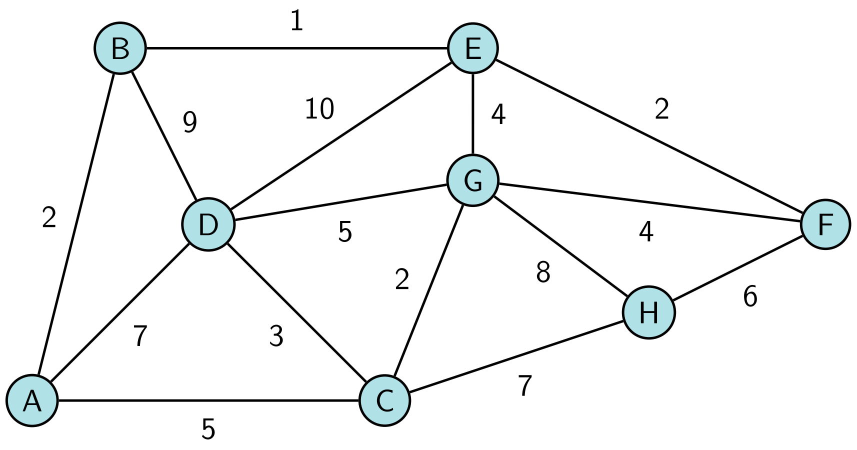
   - Quick sort n^2
   - Dijkstra's algorithm O(IEI + IVIlogIVI)
   - Heap Sort nlogn
   - Breadth-first search of a graph O(E+V)
   - Searching in a height-balanced tree. logn

4. (10 points) Determine the worst-case time complexity of the following pseudocode algorithm. Assume that `DoOtherStuff` has a time complexity of $O(n)$ where $n = r - l$. Show all calculations, justify your answer.

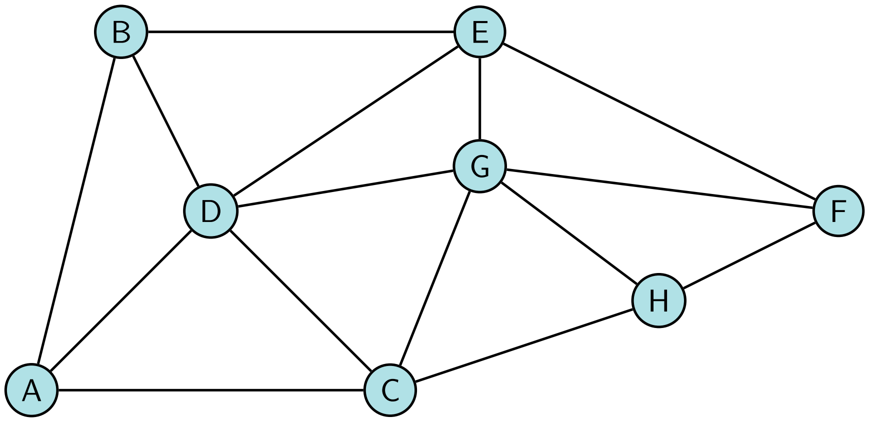
```
Algorithm DoStuff ( a, l, r )
   'a' is an array of numbers
   'l' and 'r' are positive integers between 0 and (a.length-1)

   int m = ( l + r ) / 2;
   if( l < r )
      DoStuff(a, l, m)
      DoStuff(a, m+1, r)
      DoOtherStuff(a,l,m,r)
```

5. (10 points) Dijkstras algorithm is a graph algorithm to determine the shortest path from a specified vertex to every other vertex in a weighted graph. Recall that it builds a tree of shortest paths from the specified vertex one edge at a time. For the graph below with specified **start vertex E**, give the **edges** (pairs of vertices!) added to the shortest path tree in the order that Dijkstras algorithm adds them to the tree.



6. (6 points) Give two different possible breadth-first search trees that might be obtained by a breadth first search of the following unweighted graph starting at vertex F.



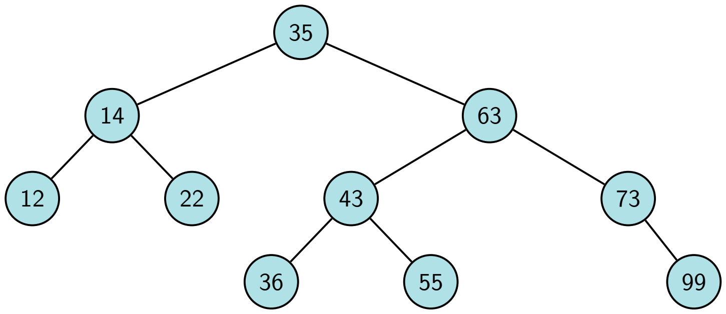
7. Consider the following problem: given an undirected weighted graph G with a single connected component (the graph is connected), find the set of edges with the smallest total cost which, if removed from the graph, results in a graph that is no longer connected. This problem is called *minimum cut*.

   (a) (5 points) Give a greedy, polynomial-time pseudocode algorithm that tries to find the minimum cut, but doesn't guarantee the optimum cut, and runs in polynomial time.

   (b) (5 points) Give an exponential-time backtracking pseudocode algorithm that is guaranteed to find the minimum cut.

8. (10 points) Suppose you are implementing a 2d-tree (a kd-tree that stores 2D points) where the nodes of the tree store objects of type `Pair280`. Assume that a `KDTree` object is implemented as a linked tree of `KDNode` objects, each storing a `Pair280` object and references to its left and right children. Give the java code, including javadoc comments, for a recursive method for performing a range search on the tree. You may use classes from the Java API or lib280 for storing the points that are found to be in range.

9. (10 points) Consider the following height-balanced tree. For each of the operations below, apply the operations to the **original** tree pictured here, and draw the resulting tree.



   (a) insert 72

   (b) delete 99, then delete 73

   (c) insert 5, then insert 8

   (d) insert 40

   (e) delete 55

10. (30 points) Design a set of classes for a graph data structure that meets the following requirements:

   - One should be able to associate any type of data item with vertices.
   - One should be able to associate any type of data item with edges.
   - The graph must have, at a minimum, the operations of:
     - Create a new graph with N nodes
     - Add an edge to the graph.
     - Delete an edge from the graph.
     - Determine if there is a path between two given nodes.
     - Retrieve the item associated with a given edge.
     - Retrieve the item associated with a given vertex.

   You may define any number of classes that you wish. Give the java code for all class definitions, including instance variables, method headers, javadoc comments for method headers, but **not** method bodies (implementations). Don't forget to document any exceptions that a method might throw. *Do not make use of classes from lib280.*

11. Answer these questions about sorting:
    (a) (7 points) Which two sorting algorithms studied in class use the *divide and conquer* approach? Compare and contrast the ways that each of these sorting algorithms subdivide the problems and combine the subproblem solutions.
    (b) (5 points) Give the pseudocode algorithm for bucket sort.

12. (20 points) Design classes for a data structure to keep track of course prerequisites at a university. A course may have any number of prerequisites, including none. A course is represented by it's subject (e.g. CMPT) and it's course number (e.g. 280). The class that stores the prerequisite information must provide the following operations:

    - Add a new course (initially with zero prerequisites)
    - Make course X as a prerequisite to course Y
    - Retrieve a list of the prerequisites for a course, including those that are not a direct prerequisite.

    You may define any number of classes that you wish. You may make use of any classes from the java API and/or lib280 that seem appropriate. Give the java code for all class definitions, including instance variables, method headers. Comment on the purpose of each instance variable so that it is clear what each one is for. Javadoc comments for methods are not necessary.